Computation and Deduction

Lecture 18: Unification

1. Types and Constraints
2. Unification Logic
3. Constraint Generation
4. Completness and Soundness

Types and Constraints

```
% Simple types
tp : type.                                  %name tp T a.
=> : tp -> tp -> tp.
%infix right 8 =>.

% Propositions of unification logic (constraints)
prop : type.                                %name prop C.

== : tp -> tp -> prop.
&  : prop -> prop -> prop.
tt : prop.
exists : (tp -> prop) -> prop.

%infix none 10 ==.
%infix right 9 &.
```

Unification Logic

```
% Truth of propositions
true : prop -> type.                          %name true P.
%mode true *C.

==I : true (T == T).
&I  : true (C1 & C2)
       <- true C1
       <- true C2.
ttI : true tt.
existsI : {T:tp} true (exists C)
                    <- true (C T).
```

Extrinsic Typing

```
% Expressions
exp : type.                                  %name exp E x.
lam : (exp -> exp) -> exp.
app : exp -> exp -> exp.

% Extrinsic typing judgment
of : exp -> tp -> type.                      %name of D.
%mode of +E *T.

of_lam : of (lam E) (T1 => T2)
          <- ({x:exp} of x T1 -> of (E x) T2).

of_app : of (app E1 E2) T1
          <- of E1 (T2 => T1)
          <- of E2 T2.
```

Constraint Generation

```
% Constraint generation
con : exp -> tp -> prop -> type.          %name con N.
%mode con +E +T -C.

con_lam : con (lam E) T (exists [a1] exists [a2]
                              C a1 a2 & T == (a1 => a2))
           <- ({x:exp} {a1:tp} ({T':tp} con x T' (a1 == T'))
                 -> {a2:tp} con (E x) a2 (C a1 a2)).

con_app : con (app E1 E2) T (exists [a2] C1 a2 & C2 a2)
           <- ({a2:tp} con E1 (a2 => T) (C1 a2))
           <- ({a2:tp} con E2 a2 (C2 a2)).

%terminates E (con E T _).
```

Checking Typability

```
con' : exp -> prop -> type.                %name con' N'.
%mode con' +E -C.

con'0 : con' E (exists [a] C a)
         <- ({a:tp} con E a (C a)).
```

Completeness

```
complete : con E T C -> of E T -> true C -> type.
%mode complete +N +D -P.

complete_lam : complete (con_lam N)
                (of_lam D : of (lam E) (T1 => T2))
                (existsI T1 (existsI T2 (&I (==I) P)))
                <- ({x:exp} {u:of x T1}
                    {n:{T':tp} con x T' (T1 == T')}
                      complete (n T1) u (==I)
                      -> complete (N x T1 n T2) (D x u) P).

complete_app : complete (con_app N2 N1)
                (of_app D2 (D1 : of E1 (T2 => T)))
                (existsI T2 (&I P2 P1))
                <- complete (N1 T2) D1 P1
                <- complete (N2 T2) D2 P2.

%terminates N (complete N D _).
```

Soundness

```
sound : con E T C -> true C -> of E T -> type.
%mode sound +N +P -D.

sound_lam : sound (con_lam N)
             (existsI T1 (existsI T2 (&I (==I) P)))
             (of_lam D)
             <- ({x:exp} {u:of x T1}
                 {n:{T':tp} con x T' (T1 == T')}
                   sound (n T1) (==I) u
                   -> sound (N x T1 n T2) P (D x u)).

sound_app : sound (con_app N2 N1)
             (existsI T2 (&I P2 P1))
             (of_app D2 D1)
             <- sound (N1 T2) P1 D1
             <- sound (N2 T2) P2 D2.

%terminates N (sound N P _).
```